# GRPC文档

## 简介

高性能、开源、通用RPC框架。

默认GRPC使用Protocol Buffer。

## Nuget添加

## ProtoBuffer

<https://developers.google.com/protocol-buffers/>

Proto不适合处理大量的数据，处理的数据级别是MB

使用.proto文件定义数据和协议格式，使用protoc生成代码

使用proto3版本。

Protocol Buffer是语言中立，平台中立的序列化反序列化库。

在.proto文件中新增字段，不会打破后向兼容性。删除字段会赋值为默认值。（注：删除字段后，不能使用原来的数字标号）。如果值是默认值，序列化时不会序列化该值。

如果消息不包含某个字段，对应的字段会赋值为默认值。默认值如下：

string使用empty string。

bytes使用 empty bytes

bool使用false

数值型使用0

* 枚举型使用第一个定义的枚举值 For [enums](https://developers.google.com/protocol-buffers/docs/proto3#enum), the default value is the **first defined enum value**, which must be 0.

Message field默认不赋值

Repeat field（序列化反序列化会保留顺序）默认empty list

对于标量类型，反序列化后无法知道使用的是默认值还是赋值的默认值0。

syntax = "proto3";

message SearchRequest {

string query = 1;

int32 page\_number = 2;

int32 result\_per\_page = 3;

enum Corpus {

UNIVERSAL = 0;

WEB = 1;

IMAGES = 2;

LOCAL = 3;

NEWS = 4;

PRODUCTS = 5;

VIDEO = 6;

}

Corpus corpus = 4;

}

每个字段必须赋值一个唯一的数字，1到15使用一个字节表示数字和类型，16到2047使用两个字节。最小的是1，最大的是 229 – 1=536,870,911。并且不能使用19000到 19999。

多个消息类型可以定义在一个.proto文件中。

//表示注释

Proto编译器根据你选择的语言生成类（包含get,set,序列化和反序列化方法）

For C#, the compiler generates a .cs file from each .proto, with a class for each message type described in your file.

**标量类型**

|  |  |  |  |
| --- | --- | --- | --- |
| proto Type | Notes | Java Type | C# Type |
| double |  | double | double |
| float |  | float | float |
| int32 | Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead. | int | int |
| int64 | Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead. | long | long |
| uint32 | Uses variable-length encoding. | int[1] | uint |
| uint64 | Uses variable-length encoding. | long[1] | ulong |
| sint32 | Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s. | int | int |
| sint64 | Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s. | long | long |
| fixed32 | Always four bytes. More efficient than uint32 if values are often greater than 228. | int[1] | uint |
| fixed64 | Always eight bytes. More efficient than uint64 if values are often greater than 256. | long[1] | ulong |
| sfixed32 | Always four bytes. | int | int |
| sfixed64 | Always eight bytes. | long | long |
| bool |  | boolean | bool |
| string | A string must always contain UTF-8 encoded or 7-bit ASCII text. | String | string |
| bytes | May contain any arbitrary sequence of bytes. | ByteString | ByteString |

importing

import "myproject/other\_protos.proto";

import public "new.proto";

import可以引用别的proto文件 ，import是不传递引用的，import public传递引用

可以使用嵌套类型Parent.ChildType,不限制嵌套的层数

当更新一个消息类型时需要注意：

1. 不要改变现有的字段的数字标签
2. 如果新增字段，则原来的格式是完全兼容的，新的协议解析旧的时，新的字段会使用默认值。原来的格式在解析新的时忽略新的字段。未知字段会被忽略
3. 删除旧的字段，要保证原来的tag number不再使用。删除的字段会在旧的协议中忽略。

Map类型使用如下语法：map不保证顺序

map<key\_type, value\_type> map\_field = N;

 key\_type can be any integral or string type (so, any [scalar](https://developers.google.com/protocol-buffers/docs/proto3#scalar) type except for floating point types and bytes). The value\_type can be any type except another map.

package foo.bar;

package包名用来解决命名冲突

In **C#** the package is used as the namespace after converting to PascalCase, unless you explicitly provide anoption csharp\_namespace in your .proto file.

定义rpc

service SearchService {

rpc Search (SearchRequest) returns (SearchResponse);

}

编译proto文件

Protoc --cpp\_out=*DST\_DIR* --java\_out=*DST\_DIR* --python\_out=*DST\_DIR* --go\_out=*DST\_DIR* --ruby\_out=*DST\_DIR* --javanano\_out=*DST\_DIR* --objc\_out=*DST\_DIR* --csharp\_out=*DST\_DIR* *path/to/file*.proto

--csharp\_out指定C#代码输出的路径

最后可以跟多个proto文件

Proto3丢弃了required，默认是optional，除非加了repeated。Proto3中repeated默认是packed的

repeated int32 d = 4

//消息名和枚举名使用CamelCase方式，如SearchRequest

//字段名使用\_分割的小写方式 如page\_number result\_per\_page

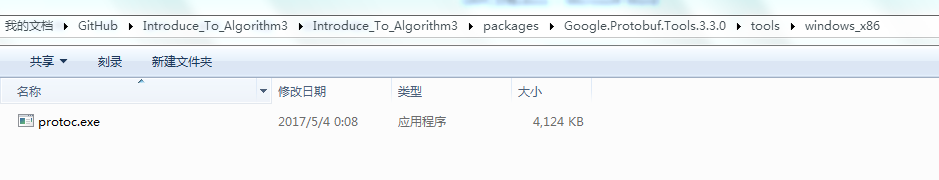
//枚举值使用 CAPITALS\_WITH\_UNDERSCORES 如 FIRST\_VALUE

//服务名和方法名使用CamelCase方式

## PB文件编译

建.net4.5工程使用Nuget安装Google.Protobuf和Google.Protobuf.Tools 和GRPC.Tools。

Google.Protobuf.Tools下放着protoc.exe和一些预定义的.proto文件

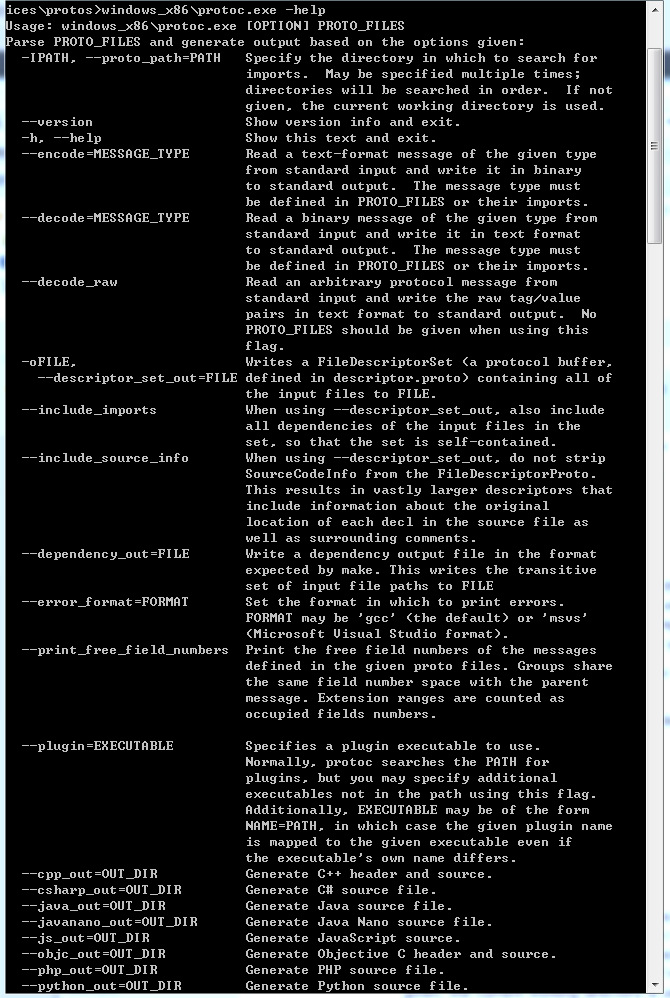




//执行命令 protoc -I=$SRC\_DIR --csharp\_out=$DST\_DIR $SRC\_DIR/addressbook.proto

//-I指定查找import的目录，默认是程序当前目录，--sharp\_out指定C#代码输出目录

//C# pb需要.net4.5

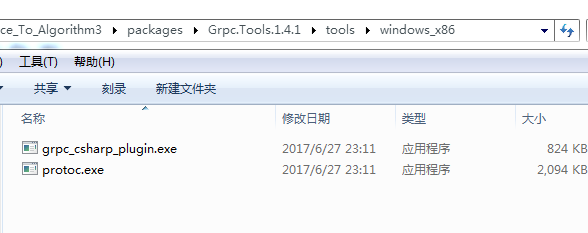


GRPC.Tools增加了对service的编译支持

## GRPC样例

第一步：编写.proto文件

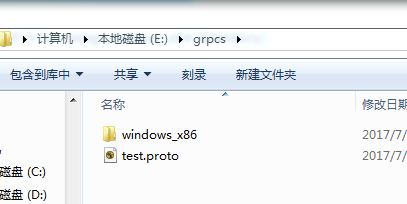
第二步：建.net4.5工程，使用Nuget下载Grpc.Tools

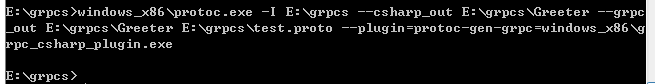


第三步：使用protoc命令编译.proto文件

第四步：使用Nuget下载Grpc，实现服务器端抽象类

第五步：绑定ip和端口，启动程序





//需要事先创建greeter文件夹

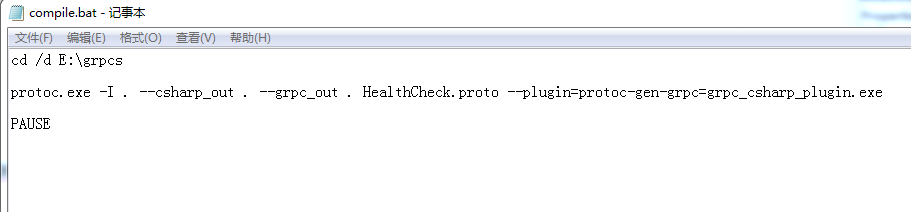
windows\_x86\protoc.exe -I E:\grpcs --csharp\_out E:\grpcs\Greeter --grpc\_out E:\grpcs\Greeter2 E:\grpcs\test.proto --plugin=protoc-gen-grpc=E:\grpcs\windows\_x86\grpc\_csharp\_plugin.exe

可以使用.表示当前目录

-I 指定搜索import的目录

--csharp\_out 生成cs序列化代码的目录

--grpc\_out 生成grpc相关代码的目录



生成两个文件Test.cs和TestGrpc.cs. Test.cs包含protocol buffer代码序列化反序列化message type。TestGrpc.cs包含客户端和服务器端代码：1)需要实现的Greeter service抽象类GreeterBase; 2)客户端类GreeterClient

整个例子打包在grpcs.zip文件下

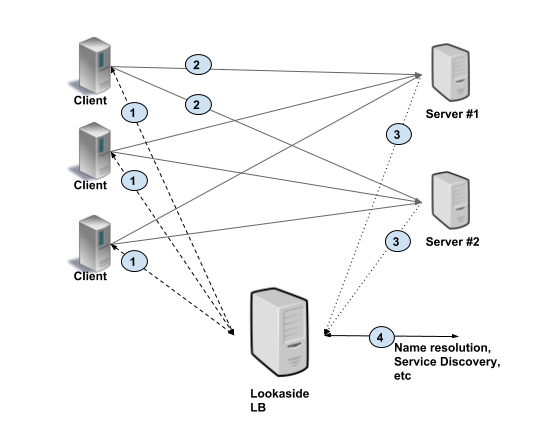


## GRPC负载均衡和监控

后台有多个相同的服务器端。客户端负责选择调用那个服务器端。

Grpc底层协议是HTTP/2。

采用Lookaside Load Balancing。其结构如下：



客户端通过询问Lookaside LB获取服务器的地址，然后调用服务器。LB负责监控服务器，移除异常的服务器。

监控端负责监控Lookaside LB。

### Lookaside LB

所有组件都在.net4.5下编写。

Lookaside LB使用sqlite作为后台数据库，LB轮询服务，检查服务状态，可以添加或者移除服务，对客户端提供服务器的ip和端口号。

监控，可选组件。Sqlite作为后台数据库。只允许连接LB，LB可以转发监控的命令到服务器，用于监控服务器和LB。

服务对客户端提供接口。